2

:UMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
FINAL 1 Apr 87 - 31 Jul 91

4. TITLE AND SUBTITLE
"TECHNIQUES FOR THE DESIGN & IMPLEMENTATION OF HIGHLY RELIABLE MULTI-PROCESSING SYSTEMS" (U)

5. FUNDING NUMBERS
61102F
2304/A2

6. AUTHOR(S)
Dr. David C. Luckham

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Stanford University
Computer Systems Laboratory
Stanford, CA 94305-4055

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 91 0923

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Building 410
Bolling AFB DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR-87-0150

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
Distribution unlimited

12b. DISTRIBUTION CODE
UL

13. ABSTRACT *(Maximum 200 words)*

Development of the Task Sequencing Language, TSL-1, for specifying ADA tasking programs was completed. Formal operational sematics were developed to allow TSL runtime monitoring tools. Loosely couple' distributed ADA systems are supported. A timing construct was developed based on partial ordering of events. Work on TSL-2, to include distributed computing and different hierarchical designs on concurrent systems, was begun and forms the basis for research on language design.

14. SUBJECT TERMS

15. NUMBER OF PAGES
10

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
SAR

STANFORD UNIVERSITY

Final Report under AFOSR Grant–87–0150C

Techniques for the Design and Implementation of Highly Reliable Multi-Processing Systems

June 1987 – July 1991

Principal Investigator: David C. Luckham

Abstract

Research undertaken under AFOSR Grant 87–0150C is described. This research focuses on specification languages for multi-processor systems, with particular emphasis on applications to Ada software. The research, however, applies generally to specifying distributed systems containing both software and hardware components, and to software systems implemented in any programming language. The primary goals are *(1)* design of a high level specification language for distributed systems, and *(2)* design and development of prototype tools for applying this language to development of highly reliable multi-processor Ada software.

This effort involved research into basic questions concerning:

- event-based models of distributed (local time asynchronous) computations,
- constraint-based concurrent specification languages,
- realtime specifications,
- methodology and support tools for specifying concurrent programs,
- implementability studies.

Over the period in question, this research effort has had technical impact in the following areas (as described in later sections):

1. Formal Ada tasking specifications
2. New Prototyping Languages
3. Industrial Tools for Process Programming
4. Training of Ph.D.'s for industry and academia.

1 Description of Research Undertaken

Under this AFOSR contract we have developed a specification language, Task Sequencing Language (TSL), for specifying Ada tasking programs. TSL is a language in which constraints on patterns of behavior of a distributed program can be expressed.

A major concept of TSL is that *patterns of events* are central in the adequate and complete specification of multi-tasking behavior (i.e., behavior involving multiple threads of control simultaneously).

The principal constructs in TSL are aimed at making it easy to describe sequences and other patterns of events in a program that is executing on many processors simultaneously. Only significant events need be used in constraints, and other events in an Ada computation can be omitted. Important events in a multi-tasking program may include, for example, communication and synchronization events between separate threads of control (e.g., rendezvous events in Ada programs). Events of importance can then be combined by TSL specification constructs to define required (or erroneous) patterns of behavior.

A second major concept of TSL is *runtime checkability*. TSL is designed so that computationally feasible algorithms exist for checking consistency between a TSL specification and a distributed computation at runtime. The rationale for checkability is to develop a wide range of applications to concurrent systems, from specification and requirements analysis, to testing, debugging, and self-checking. In view of the difficulty of verifying correctness of concurrent systems, and the absence at present of automated proof systems for concurrency, self-checking distributed systems is a promising practical approach to the pressing problems of reliability and security.

The emphasis on checkability does not, however, conflict with formal consistency proof applications. In fact, checking and proof should be regarded as complementary techniques with a common basis, namely, both methods utilize the same TSL specifications. Consistency proof methods can be applied to TSL specifications whenever proof rules are developed. (Proof rules for the most recent version of TSL are currently being developed.)

During 1987, the design of TSL-1 was completed. TSL-1 is based on an observational model of a distributed computation as a linear stream of events. TSL-1 implementations support both the specification and testing phases of tightly coupled concurrent Ada systems. Basic concepts of TSL-1 have been deployed in SA/PDL, an Ada-based simulation language developed by IDA for the SDI [12].

Support tools have been designed for TSL-1, including a preprocessor that instruments Ada programs to enable tracing of tasking events, and a runtime monitor for checking consistency between the runtime behavior of Ada tasking programs and TSL-1 pattern specifications. Preliminary experimental implementations of these tools have been completed. Our experiments and publications show that the TSL-1 runtime monitor is a very powerful debugging tool for Ada tasking programs, and can detect subtle error situations such as communication protocol errors and race conditions.

Standard debugging tools are essentially useless for detecting such errors in a multiple processor environment. This has been confirmed in class exercises to debug Ada programs at Stanford. Errors in distributed systems are far too difficult to detect and reproduce by old fashioned information gathering after the error. Instead, we have pursued the alternative of using TSL-1 for specifying error behavior patterns. Violations of such specifications can be automatically detected by our runtime monitor tools as they occur.

During 1988-89 we defined a formal operational semantics for TSL-1 [10]. This report can be used as an implementation guide in constructing runtime monitoring tools for TSL or similar pattern-constraint languages.

We have completed implementation of a pilot TSL-1 toolset on a multi-processor Sequent Symmetry. This included an experimental TSL-1 runtime monitor for detecting inconsistencies between the actual behavior of a distributed Ada system and TSL-1 specifications of the behavior.

We have formalized the complete Ada tasking semantics ([1, §9]) in TSL-1 [24].

We have experimented with using the formal Ada tasking semantics as the TSL specification for an Ada tasking scheduler running on a Sequent Symmetry multi-processor. Experiments were performed that utilized this specification together with the TSL monitor, as a testbed for Ada schedulers [24,27].

We have developed a theory of interference by runtime monitors on distributed computations being monitored. We have shown that the TSL-1 monitor does not interfere with the underlying computation in ways that preclude concurrency errors from showing up, or introduce new concurrency errors. A report on these results has been written [9].

Also during 1988-89 TSL-1.5 was developed from TSL-1 to provide a more powerful specification language suitable for loosely coupled distributed Ada systems. An underlying computational model of partially ordered sets of events was adopted in place of the previous linear stream model of TSL-1. Models using partial orderings are being adopted generally in research on concurrency [23]. Semantics based on partial order models allow TSL-1.5 to express causality and timing (an event causes another event if the two events are ordered), as well as concurrency (two events are concurrent if they are not ordered). Ada tasking computations on multi-processors conform to the partial order model. A draft report on TSL-1.5 was developed [16].

Lastly, in 1988-89, further development of TSL tools and experimental applications were undertaken and published.

During 1989-90, the partial ordering model was applied to Ada. The partial ordering of events informally defined by the Ada Standard [1] was formalized and published [5].

That year, a timing construct was added to the TSL specification language. Like causality, timing is modeled as a partial ordering of events (i.e., simultaneous events are unordered).

Also in 1989–90, the prototype implementation of the TSL-1 preprocessor and runtime monitor was completed and a simple, interactive, source-level debugger was built into the monitor. To support the tools, a 73 page users' guide was developed [4]. (These tools are now available on the Internet.)

Initial work on pattern-based *mappings* as a means of expressing abstraction was begun during 1989-90 and preliminary results were published [20].

During 1990–91, design of a new machine processable specification language, TSL-2, for distributed systems was begun. TSL-2 is evolved from previous project work on the design and implementation of TSL-1 and the design of TSL-1.5. TSL-2 adds a few new constructs to TSL and extends the semantics of the existing ones to enable specification of the most general forms of distributed computation, and also to specify hierarchical designs of concurrent systems. In particular, this work has involved:

- In TSL-2 the basic pattern language for expressing constraints has been improved to provide features for specifying causality between events, timing, overlapping events, and independence of events in distributed systems.
- The semantics of TSL-2 is defined using partial order models of distributed computation.
- A facility for *behavioral abstraction* was added. New constructs (not in previous versions of TSL) for expressing hierarchical decomposition of distributed systems were developed. This facility is based on the concept of pattern mappings for expressing relationships between different levels of specifications. Pattern mappings are very similar to pattern constraints so the complexity of the specification language TSL-2 is not increased.

Beginning in 1990, basic algorithms for efficient reproduction of partially ordered distributed computations were researched, a report was written [21], and a prototype implementation is underway. A model for reproducing the partial ordering of distributed computations has been worked out by Fidge [7] and Mattern [18]. (An implementation of the general Fidge-Mattern model has been completed.) We have developed algorithms to implement the Fidge-Mattern model which we believe will overcome the computational complexity of the general model. Such algorithms are critical in implementing tools to check consistency of distributed computations with TSL-2 specifications.

A first application of pattern mappings, to the monitoring of VHDL simulations, was implemented [8]. This will permit a user of TSL-2 to specify a level of detail (e.g., instruction set level, register transfer level, gate level) at which the simulation output is to be mapped and analyzed for consistency with specifications.

Also during 1990–91, TSL–2 was applied in new research on language design. TSL–2 became the basis for a concurrent specification sublanguage in a new language design effort to support prototyping. This use of TSL both improved the expressiveness of time-sensitive specifications and strengthened the theory of event patterns.

Lastly, new benchmark example applications of TSL tools, illustrating techniques for applying TSL–2 to air traffic control problems are being developed, and the toolset was extended by porting it to new host architectures. Distribution of the runtime monitor itself was investigated.

2 Application of TSL in other research efforts

In this section we give a short list of other research efforts and projects in which TSL is being used and its concepts are being applied.

1. **Ada Performance Measurements** TSL–1 tools are the subject of a proposed subcontract from Encore Inc. as part of the DARPA strategic computing initiative. It is proposed that Stanford port the TSL–1 tools to the Encore Multimax computer project and reengineer them to enhance and generalize the Encore Parasight multi-processor performance measurement facility.

2. **Ada Environment Tools** TSL–1 monitoring and debugging tools and TSL–2 specification analysis tools are a planned component of the integrated analysis tools in the DARPA Arcadia environment effort.

3. **Concurrent Program Monitors** A distributed implementation of TSL–1.5 is being developed at the University of Bergen in Norway.

4. **CAD Tools** TSL–2 hierarchical mapping constructs have recently been incorporated into a specification language for hardware design, called VAL [2], associated with the VHSIC project's VHDL. Mappings form the basis for a prototype implementation of a VHDL support tool for comparative validation of VHDL simulations.

5. **Ada Tasking Specifications** TSL–1 has been used by Mitre Corp. in the design and specification of a user interface [6]. More recently Mitre is proposing projects in software engineering environments in which TSL is a component technology for concurrency specifications.

6. **Object-Oriented Concurrency Specifications** The possibility of using TSL–2 as the concurrency specification sublanguage in the European ESPRIT project, Dragoon [17], in place of Deontic Logic, has been proposed by the project leader, Prof. S. Crespi-Regizzi of Politechnico, Milan; we have not received any firm decision as yet.

7. **Software Process Specifications** TSL–1 concepts are being applied to industrial problems at AT&T Bell Labs. Particular applications undertaken by D. S. Rosenblum of AT&T include:

- development of experimental tools for monitoring operating system-level events for consistency with event pattern specifications. For each specification, a specified action is taken whenever the specified event pattern occurs. In a current prototype implementation, the events of interest are Unix-level events such as file modification, passage of time, users logging in, etc., and the action component is a Unix command sequence [11,26].
- possible development of high-level event-pattern specification monitoring of telephone networks, replacing the current system-level event monitoring capabilities.

8. **Prototyping Languages** TSL–2 has had a significant impact in the area of prototyping languages. A new programming and design language is being designed by the Stanford/TRW team during phase-1 of the DARPA initiative on "New Language for Rapid Construction of Software Prototypes" [3,21,22]. Major features of the language include: object-oriented type model, first-order logic specifications, concurrency specifications, and pattern-based process invocation. TSL–2 has been adopted for use in both concurrency specifications and pattern-based process invocation. This use of TSL constructs has resulted in further development of TSL's pattern and specification features:

   - The ability to describe time-critical and time-sensitive computations was improved.
   - Specifications are currently being developed into an algebra including theories of checkability, substitutability and proof rules.

3 Presentations of TSL and TSL research results

1. D.C. Luckham, Tri-Ada '88 International Conference, Special Session on Innovative Ada Technology, October 1988.

2. D.C. Luckham, Formal Methods Workshop 1989, Halifax, Nova Scotia, July 1989.

3. S. Meldal, *Specifying and Observing Concurrent Programs*, Third International Workshop on Large Grain Parallelism, Carnegie Mellon University, Pittsburgh, October 1989.

4. D.L. Bryan, *An Algebraic Specification of the Partial Orders Generated by Concurrent Ada Computations*, presented at Tri-Ada '89 International Conference, October 1989 [5].

5. D.S. Rosenblum and D.C. Luckham, *Testing the Correctness of Tasking Supervisors with TSL Specifications*, presented at the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification (TAV3), December 1989 [27].

6. D.C. Luckham, Invited Lectures on Rigorous Methods in Software Engineering, Software Engineering Institute, Carnegie Mellon University, April 1990.

7. D.C. Luckham, ACM International Workshop on Formal Methods, invited lecture, "Compromises and New Directions in Formal Methods", Napa, Calif., May 1990.

8. S. Meldal, *Supporting Architecture Mappings in Concurrent Systems Design*, Australian Software Engineering Conference, Sydney, May 1990 [20].

9. F. Belz and D.C. Luckham, *A New Approach to Prototyping Ada-Based Hardware/Software Systems*, presented at Tri-Ada '90 International Conference, December 1990 [3].

10. J. Mitchell, S. Meldal and N. Madhav, *An Extension of Standard ML Modules with Subtyping and Inheritance*, presented at the ACM Conference on the Principles of Programming Languages, January 1991 [22].

11. D.S. Rosenblum, *An Overview of TSL, A Language for Specifying and Debugging Concurrent Programs*, IEEE Software, May 1991 [25].

12. S. Meldal, S. Sankar and J. Vera, presentation at the Tenth Annual ACM Symposium on Principles of Distributed Computing, *Exploiting Locality in Maintaining Potential Causality*, August 1991 [21].

References

[1] *The Ada Programming Language Reference Manual.* US Department of Defense, US Government Printing Office, February 1983. ANSI/MIL-STD-1815A-1983.

[2] Larry M. Augustin, David C. Luckham, Benoit A. Gennart, Youm Huh, and Alec G. Stanculescu. *Hardware Design and Simulation in VAL/VHDL.* Kluwer Press, October 1990. 322 pages.

[3] Frank Belz and David C. Luckham. A new approach to prototyping Ada–based hardware/software systems. In *Proceedings of the ACM Tri-Ada Conference*, ACM Press, Baltimore, December 1990.

[4] D. Bryan. A tool for specifying, checking and debugging Ada tasking programs. January 1990. Program Analysis and Verification Group, Stanford University, internal document. 73 pages.

[5] Douglas L. Bryan. An algebraic specification of the partial orders generated by concurrent Ada computations. In *Proceedings of Tri-Ada '89*, pages 225–241, ACM Press, October 1989.

[6] C.M. Byrnes. *The Application of Anna and Formal Methods as an Ada Program Design Language.* Technical Report ESD–TR–86–276, MTR–10067, The MITRE Corporation, Bedford, MA, October 1986.

[7] C.J. Fidge. Partial orders for parallel debugging. In *Workshop on Parallel and Distributed Debugging*, pages 183–194, ACM SIGPLAN/SIGOPS, Madison, Wisconsin, May 5-6, 1988.

[8] B.A. Gennart. *Automated Analysis of Discrete Event Simulations Using Event Pattern Mappings.* PhD thesis, Stanford University, April 1991. Also Stanford University Computer Systems Laboratory Technical Report No. CSL–TR–91–464.

[9] David P. Helmbold and Douglas L. Bryan. *Design of Run Time Monitors for Concurrent Programs.* Technical Report CSL–TR–89–395, Computer Systems Laboratory, Stanford University, October 1989.

[10] D.P. Helmbold. *The Meaning of TSL: An Abstract Implementation of TSL-1.* Technical Report CSL–TR–88–353, Computer Systems Laboratory, Stanford University, March 1988. Also published by Computer Information Sciences Board, UC Santa Cruz as UCSC–CRL–87–29.

[11] B. Krishnamurthy and D.S. Rosenblum. An event-based model of computer-supported cooperative work: design and implementation. In *CSCW*, pages 132–145, IFIP, 1991. Proceedings of the International Workshop on CSCW, Berlin, Germany, April 9-11.

[12] J. Linn, C. Ardion, C. Linn, S. Edwards, M.Kappel, and J. Salasin. *SDI Architecture Dataflow Modeling Technique: Version 1.5.* Technical Report IDA Paper P-2035, Institute for Defense Analysis, April 1988.

[13] D. C. Luckham, D. P. Helmbold, S. Meldal, D. L. Bryan, and M. A. Haberler. TSL: task sequencing language for specifying distributed Ada systems: TSL–1. In Habermann and Montanari, editors, *System Development and Ada, proceedings of the CRAI workshop on Software Factories and Ada. Lecture Notes in Computer Science. Number 275*, pages 249–305, Springer-Verlag, May 1986.

[14] D.C. Luckham, D.P. Helmbold, D.L. Bryan, and M.A. Haberler. Task sequencing language for specifying distributed Ada systems. In *PARLE: Parallel Architectures and Languages Europe*, pages 444–463, Springer-Verlag, 1987. Proceedings on Parallel Architectures and Languages Europe, Eindhoven, The Netherlands, June 1987.

[15] D.C. Luckham, D.P. Helmbold, S. Meldal, D.L. Bryan, and M.A. Haberler. Task sequencing language for specifying distributed Ada systems. In *System Development and Ada*, pages 249–305, Springer-Verlag, 1987. Springer-Verlag Lecture Notes in Computer Science, No. 275, Proceedings of the CRAI Consorzio per le Ricerche e le Applicazioni di Informatica Workshop on Software Factores and Ada, Capri, Italy, May 1986.

[16] D.C. Luckham, S. Meldal, D.P. Helmbold, D.L. Bryan, and W. Mann. *An Introduction to Task Sequencing Language, TSL 1.5.* Technical Report 38, Department of Informatics, University of Bergen, Bergen, Norway, August 1989. Preliminary version.

[17] A. Di Maio, C. Cardigno, R. Bayan, C. Destombes, and C. Atkinson. DRAGOON: an Ada-based object oriented language for concurrent, real-time distributed systems. In *Systems Design with Ada: Proceedings Ada-Europe International Conference*, Cambridge Press, Madrid, June 1989.

[18] F. Mattern. Virtual time and global states of distributed systems. In M. Cosnard, editor, *Proceedings of Parallel and Distributed Algorithms*, Elsevier Science Publishers, 1988. Also in: Report No. SFB124P38/88, Dept. of Computer Science, University of Kaiserslautern.

[19] S. Meldal, D.C. Luckham, and M.A. Haberler. Specifying Ada tasking using patterns of behavior. In B.D. Shriver, editor, *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, pages 129–134, ACM, January 1988.

[20] Sigurd Meldal. Supporting architecture mappings in concurrent systems design. In *Proceedings of Australian Software Engineering Conference*, IREE Australia, May 1990.

[21] Sigurd Meldal, Sriram Sankar, and James Vera. Exploiting locality in maintaining potential causality. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 231–239, ACM Press, Montreal, Canada, August 1991. Also Stanford University Computer Systems Laboratory Technical Report No. CSL–TR–91–466.

[22] John Mitchell, Sigurd Meldal, and Neel Madhav. An extension of standard ML modules with subtyping and inheritance. In *Proceedings of the ACM conference on POPL 1991*, ACM Press, January 1991. Also Stanford University Computer Systems Laboratory Technical Report No. CSL–TR–91–472.

[23] E. Odijk, M. Rem, and J-C Syre. *PARLE89: Parallel Architectures and Languages Europe.* Volume 365 of *Lecture Notes in Computer Science*, Springer-Verlag, New York, June 1989.

[24] D. S. Rosenblum. *Design and Verification of Distributed Tasking Supervisors for Concurrent Programming Languages.* PhD thesis, Stanford University, March 1988. Also Stanford University Computer Systems Laboratory Technical Report No. CSL–TR–88–357.

[25] David S. Rosenblum. Specifying concurrent systems with TSL. *IEEE Software*, 8(3):52–61, May 1991.

[26] D.S. Rosenblum and B. Krishnamurthy. An event-based model of software configuration management. In *Software Configuration Management*, pages 94–97, ACM SIGSOFT, 1991. Proceedings on the 3rd International Workshop on Software configuration Management, Trondheim, Norway, June 12-14.

[27] D.S. Rosenblum and D.C. Luckham. Testing the correctness of tasking supervisors with TSL specifications. In *Proceedings of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification (TAV3)*, pages 187–196, ACM Press, December 1989.